(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A NEW BRANCH-AND-BOUND PROCEDURE
FOR COMPUTING OPTIMAL SEARCH PATHS

by

Gustavo H. A. Martins

March 1993

Thesis Advisor: James N. Eagle
Co-Advisor: Craig W. Rasmussen

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1993	Master's Thesis

4. TITLE AND SUBTITLE
A NEW BRANCH-AND-BOUND PROCEDURE FOR COMPUTING OPTIMAL SEARCH PATHS

5. FUNDING NUMBERS

6. AUTHOR(S)
MARTINS, Gustavo H. A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We consider the problem of a searcher trying to detect a target that moves among a finite set of cells, C = 1,...,N, in discrete time, according to a specified Markov process. In each time period the searcher chooses one cell to search. Suppose the searcher is in cell j at time t. If the target is in j, it is detected with probability p_j. If the target is not in j, no detection will occur in that time period. The set of cells the searcher can choose in time t+1 is denoted C_j. If T periods of time are available for search, the searcher's objective is to maximize the probability of detecting the target during the T searches.

We propose and implement a branch-and-bound procedure for solving the problem above, using the expected number of detections as the bound. We also propose and implement a combination of two heuristic as an effective way of obtaining approximate solutions in polynomial time.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Optimal Search Paths, Search, Branch-and-Bound, Optimal Search, Moving target.	64
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited.

A NEW BRANCH-AND-BOUND PROCEDURE FOR COMPUTING OPTIMAL SEARCH PATHS

by

Gustavo H. A. Martins
LieutenantCommander, Brazilian Navy
B.S., Brazilian Naval Academy, 1980

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1993

Author: _____
Gustavo H. A. Martins

Approved by: _____
James N. Eagle, Thesis Advisor

_____
Craig W. Rasmussen, Co-Advisor

_____
Alan R. Washburn, Second Reader

_____
Peter Purdue, Chairman
Department of Operations Research

ABSTRACT

We consider the problem of a searcher trying to detect a target that moves among a finite set of cells, C = 1,..., N, in discrete time, according to a specified Markov process. In each time period the searcher chooses one cell to search. Suppose the searcher is in cell j at time t. If the target is in j, it is detected with probability p_j. If the target is not in j, no detection will occur in that time period. The set of cells the searcher can choose in time t+1 is denoted C_j. If T periods of time are available for search, the searcher's objective is to maximize the probability of detecting the target during the T searches.

We propose and implement a branch-and-bound procedure for solving the problem above, using the expected number of detections as the bound. We also propose and implement a combination of two heuristics as an effective way of obtaining approximate solutions in polynomial time.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. PROBLEM DEFINITION

A searcher and target move in discrete time and space. The number of time periods is T and the number of positional cells is N. The target's initial position is given by $\pi(1)$, a probability distribution over the N cells. The searcher's initial cell is specified. In each time period, one cell is searched and, after the search, there is a target transition, following a specified Markov transition matrix, Γ.

Suppose the searcher moves to cell j. If the target is in j, it will be detected with probability p_j. If the target is not in cell j, no detection will occur in that time period. We define C_j as the set of cells that the searcher can reach in the next time period.

We choose as our Measure of Effectiveness (MOE) the probability of detecting the target (PD) in T time periods, and define our problem as finding a feasible path that, if followed, will maximize this probability.

An efficient procedure for solving this problem was proposed by Eagle and Yee [Ref. 1]. This method uses a branch-and-bound algorithm, with the bounds computed using the Frank-Wolfe Method [Ref. 2] to solve a relaxed nonlinear program. This procedure was investigated by Caldwell [Ref. 3] in his

master's thesis, where he identified the importance of a good initial solution [p. 28] and observed the long running times associated with larger problems [pp. 45-47].

In this thesis, we propose a new heuristic for obtaining fast "good" initial solutions and a new bound, which is easy and fast to compute, but is not as tight as the one used by Eagle and Yee.

In Section B we will discuss the Eagle-Yee branch-and-bound procedure.

B. DISCUSSION OF EAGLE-YEE PROCEDURE

A network is composed of NxT nodes, as in Figure 1, where each node represents one cell in a given time period. We use arcs to represent feasible moves for the searcher. Therefore,

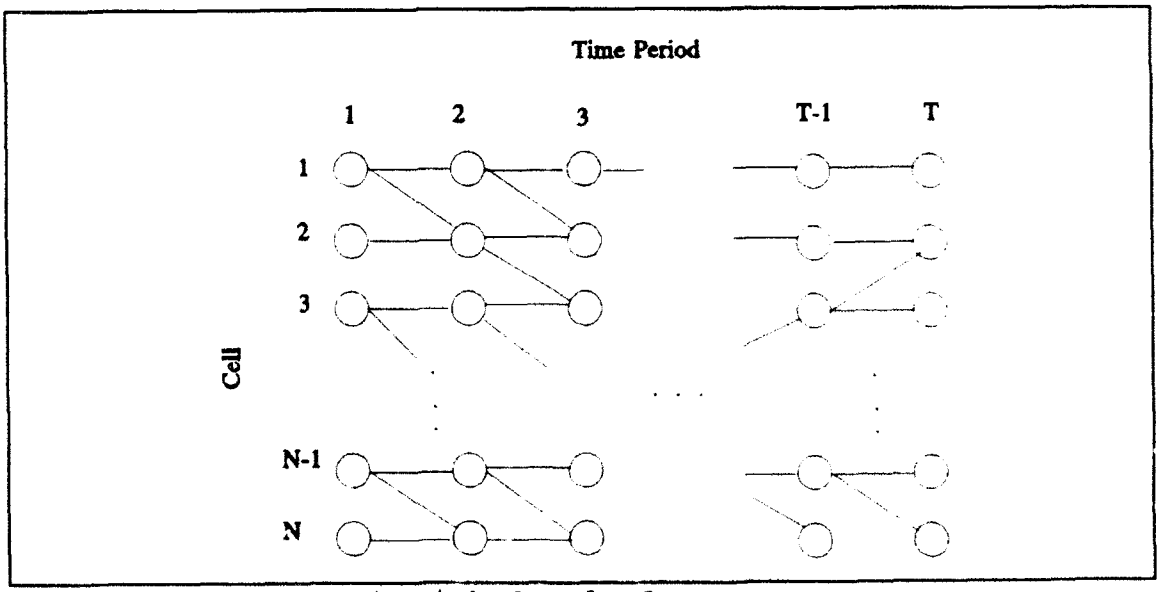

Figure 1 NxT Network with Sample Arcs

if the searcher is in cell j at time t, a set of directed arcs will connect this node to the nodes in C_j at time t+1. The network search problem is to find a feasible path through the network which minimizes the probability of nondetection.

Eagle and Yee used a branch-and-bound solution procedure, with the bound calculated by solving a "relaxed problem" (RP) using the Frank-Wolfe method. In the RP, the integrality of the searcher position is relaxed. Specifically, let x(i,j,t) be the search effort flow from cell i at time t to cell j at time t+1. The total search effort in cell j at time t then is

$$X(i,j) = \sum_{i|j \in C_i} x(i,j,t-1) .$$

We also assume an exponential detection function; that is, the probability of detection in cell j, given the target is in that cell is $1 - \exp(-\alpha_j X(j,t))$. Here α_j is the detection rate in cell j, and is such that $p_j = 1 - \exp(-\alpha_j)$. Let Ω be the set of all feasible target paths and ω be a path in this set. We denote $\omega(t)$ as the cell occupied by the target at time t.

The RP, as stated by Eagle and Yee [Ref. 1, pp. 111] is: Minimize

$$\sum_{\omega \in \Omega} p_{\omega} \exp\{-\sum_{t=1}^{T} \alpha_{\omega(t)} \sum_{(i \mid \omega(t) \in C_i)} x(i, \omega(t), t-1)\}$$

Subject to

$$X(i,0) - \sum_{j \in C_j} x(i,j,0) = 0, \; i=1,\ldots,N$$

$$\sum_{(i \mid j \in C_i)} x(i,j,t-1) - \sum_{k \in C_j} x(j,k,t) = 0, \; j=1,\ldots,N; \; t=1,\ldots,T-1$$

$$x(i,j,t) \geq 0, \; i,j=1,\ldots,N; \; t=0,\ldots,T-1.$$

The Eagle-Yee procedure uses the fact that the feasible region for the relaxed problem is convex, with linear network constraints, and computes a lower bound to the solution of the relaxed problem (RP) at each branch decision, using the Frank-Wolfe method.

The Frank-Wolfe method for this problem proceeds by solving a converging sequence of shortest path problems, where the gradient of the objective function is the value to be minimized.

The major advantage of the bounding procedure suggested herein, is that each bound calculation requires the solution

of a single longest path problem. This new bounding procedure will be explained in Chapter II.

II. THE NEW BOUND

As stated in Chapter I, the idea of modeling this problem as a network, using a branch-and-bound algorithm with relaxed constraints for the bound, is not new but the approach given here uses a different bound that is easier to calculate. Stewart [Ref. 4] relaxed the searcher's path constraints to obtain a lower bound on PND. Eagle and Yee maintained the path constraints, but relaxed the binary condition on the searcher's position. We will maintain both the searcher's path and the binary constraints, but will bound the probability of detection (PD) above by the expected number of detections.

A. CALCULATING AN UPPER BOUND ON PD

We start this section with some definitions, in addition to those from Chapter I. Let **S** denote the set of feasible search paths. Let s be a path in **S**. The cell being searched at time t is identified as s(t). Let $Q_s(j, m)$ be the probability of initial detection during times j,...,m when searcher follows path s. Let D_s be the number of detections occurring during the search. Also, define $P(D_s = k)$ as the probability of k detections when the searcher follows path s.

If T time periods are available for search, our problem is finding PD = Q(1,T), where

$$Q(1,T) = \max_{s \in S} Q_s(1,T) . \qquad \textbf{(eqn 2.1)}$$

One way of representing $Q_s(1,T)$ is by

$$Q_s(1,T) = \sum_{k=1}^{T} P(D_s=k) . \qquad \textbf{(eqn 2.2)}$$

One important property of $Q_s(1,T)$ that will be used later in this section is that the event of initial detection must occur either in periods 1,...,m or m+1,...,T. So,

$$Q_s(1,T) = Q_s(1,m) + Q_s(m+1,T) . \qquad \textbf{(eqn 2.3)}$$

The difficulty with using PD as the objective function is that the choice of the cell to be searched at each time period may depend on the searcher path up to that point. More specifically, it depends on the current target distribution, given all the search to date.

To address this difficulty, we solve the related but simpler problem of finding the path that maximizes the expected number of detections, E[D], as in Eagle and Washburn [Ref. 5]. The relation between these two problems becomes apparent when we observe that

$$E[D_s] = \sum_{k=1}^{T} kP(D_s=k) \geq \sum_{k=1}^{T} P(D_s=k) = Q_s(1,T), \qquad \textbf{(eqn 2.4)}$$

where $E[D_s]$ is the expected number of detections obtained when the searcher follows path s. So $E[D_s]$ is an upper bound on $Q_s(1,T)$ and since this is true for all paths $s \in S$, it is also true that

$$E[D] \equiv \underset{s \in S}{Max}\ E[D_s] \geq \underset{s \in S}{Max}\ Q_s(1,T) = Q(1,T). \qquad \textbf{(eqn 2.5)}$$

That is, E[D] is an upper bound on Q(1,T).

The problem of finding the path s maximizing $E[D_s]$ is simpler than finding the path u maximizing $Q_u(1,T)$. This is the case since the solution for the first problem can be obtained using a longest path algorithm, as we will explain in the next section.

What we need now is to understand how to use E[D] to solve our problem. Suppose that our way of finding an optimal search path consists of trying all possible paths, keeping record of those giving the best results, and $Q_u(1,T)$, for some $u \in S$, is the best result so far. We, also, have been following path s during times 1,...,m. We would like to know

if there exists an extention path s'(t), for t = m+1,...,T, such that

$$Q_u(1,T) \leq Q_s(1,m) + Q_{s'}(m+1,T) . \qquad \textbf{(eqn 2.6)}$$

If such extention path s' does not exist, we can guarantee that any T time period path that is coincident with s, during times 1,...,m, will give us a worse result than path u. So we can discard all such paths. As an example, if m = 5, T = 10 and at each time period the searcher can choose 5 different search cells, we are discarding $5^5 = 3125$ paths.

The nonexistence of s' can be determined by maximizing the probability of detection during times m+1,...,T, finding Q(m+1,T) and comparing the result with $Q_u(1,T) - Q_s(1,m)$. But this maximization problem is still hard to solve, so we rely again on our simplification and find the path s* that maximizes the expected number of detections during times m+1,..., T, with no detections occurring during times 1,...,m, obtaining $E[D_{s*}]$, which is an upper bound on Q(m+1,T). Then if

$$Q_u(1,T) > Q_s(1,m) + E[D_{s*}] , \qquad \textbf{(eqn 2.7)}$$

we can discard all T time period paths that are coincident with s from time period 1 though m.

B. MAXIMIZING THE EXPECTED NUMBER OF DETECTIONS

In Section A we mentioned that the path maximizing the expected number of detections can be obtained via a longest

path algorithm. In this section we show how this is accomplished.

Define P(t) as the target probability distribution at time t, where P(t,j) is the probability that the target is in cell j at time t. This distribution is obtained by the formula

$$P(t) = \pi(1) \cdot \Gamma^{t-1}; \qquad \textbf{(eqn 2.8)}$$

We can state our Longest Path Problem as finding the path through the NxT node network that yields the maximum total reward, where the reward p_j * P(t,j) is received when the searcher visits cell j at time t. Then

$$Z \equiv \underset{s \in \mathcal{S}}{Max} \sum_{k=1}^{T} p_{s(t)} * P(k, s(k)) . \qquad \textbf{(eqn 2.9)}$$

Let Y(t), t = 1,..., T be indicator random variables that take value 1, if detection occurs at time t, or 0 otherwise. Then the number of detections during 1,..., T when following path s is equal to the sum of the Y(t),

$$D_s = \sum_{t=1}^{T} Y(t) . \qquad \textbf{(eqn 2.10)}$$

Therefore, the expected number of detections is equal to the sum of the expected values of Y(t). But since Y(t) takes value 1 with probability $p_{s(t)}$ * P(t, s(t)), we can see that Z, the

solution of equation (2.9) is the maximum expected number of detections.

Now, let $\pi(t)$ be the target undetected probability mass at time t. We use this notation since the undetected probability mass at time 1 is the initial distribution of the target, $\pi(1)$. Then, if the searcher follows path s,

$$\pi(t+1) = \pi(t) \cdot M_{s(t)} \cdot \Gamma, \qquad \textbf{(eqn. 2.11)}$$

where $M_{s(t)}$ is a diagonal matrix with 1 for all diagonal entries, except for the "s(t)" entry, which takes value $1-p_{s(t)}$. Now , if the searcher followed path s through time k, compute P(t) by

$$P(t) = \pi(k) \cdot \Gamma^{t-k}, \quad t \geq k. \qquad \textbf{(eqn 2.12)}$$

P(t, j) now gives the probability that the target is in cell j at time t and was not detected during times 1,...,k. So, we can compute the maximum expected number of detections for times k+1 to T, with no detections occurring during times 1,...,k, using the same approach as before.

Further discussion on which Longest Path Algorithm to use will be covered in Chapter III.

Now, it is necessary to study a way of finding a "good" initial solution, since the better this solution is, the more efficiently will the procedure perform.

C. A HEURISTIC TO FIND "GOOD" SOLUTIONS

Another point that requires attention in a branch-and-bound procedure, as in most iterative processes, is how to obtain an initial solution. Usually, but not always, the closer to the optimal solution we start, the faster the procedure will perform. Therefore, it would be interesting to be able to find a path yielding a high PD, if we could do it fast.

One would initially suggest the path maximizing E[D] as a natural candidate, since we will be finding this path any way, and it seems reasonable to suppose that it would guarantee a high PD. This may be true in some situations, but not always, as we will see in an example.

Example 2.1: Suppose we have a three-cell problem and a stationary target that is in cell 1, with 34% probability, and in cells 2 or 3, with 33% probability each. Let $p_j = 1.0$ in all cells. We also assume there are three time periods available and the searcher can choose any cell to search in each period. The path maximizing $E[D_s] = 1.02$ is (1,1,1), with PD = 0.34, while one of the paths maximizing PD would be (1,2,3), with PD = 1.0.

The example above shows us the deficiency associated with using $E[D_s]$ as a criterion in choosing a path: if we want to maximize $E[D_s]$, we will try to search in those cells with

higher PD, without considering the fact that we might have searched there before. But we can still use this idea in an heuristic. In this heuristic we update the target undetected distribution after the search in each time period. The pseudocode for the heuristic is as follows:

```
PROCEDURE Heuristic_1;
1 Path(0) := Initial cell of searcher;
2 FOR t := 1 TO T
3     Let Path(t-1) be the searcher's cell;
4     Find extention path s' maximizing E[D_t] for times
      t,...,T, given no detection during times 1,...,t-1;
5     Path(t) := s'(t);
6     Compute π(t+1) from π(t);
7 Compute PD following Path;
8 Return Path and PD.
```

Since this procedure updates, in each time period, the target distribution, given that no detection has occurred during the previous search, we would expect better results. If we apply the *Heuristic_1* procedure to the problem in example 2.1, we would obtain *Path* = (1,2,3) and *PD* = 1.0.

It is interesting to observe that if at line 4 of the pseudocode, instead of maximizing $E[D_t]$ for times t,...,T, given no detection during times 1,...,t-1, we were maximizing

$Q_i(t-;t)$, the probability of first detection at time t, we would have the myopic search [Ref. 6].

The result of this heuristic applied to different problems will be discussed in Chapter IV.

III. IMPLEMENTATION

The appendix includes the program Find_Path, which is our computer implementation of the algorithm. This program was developed using the Pascal programming language. In this Chapter, the pseudocode for the different functions and procedures used will be presented.

A. THE LONGEST PATH ALGORITHM

The Longest Path Algorithm used in this implementation was adapted from Cormen, Leiserson and Rivest [Ref. 7]. Since the network used to model our problem has arcs that can be traversed only in the direction of increasing time, it is a directed acyclic graph (DAG), and we choose the DAG Longest Path Algorithm [Ref. 7, p. 536] to compute the longest path at each branch decision of the branch-and-bound procedure.

The basic technique in this algorithm is called **relaxation**. Let *IniDist* = $Q_s(1,k-1)$, where s is the path followed by the searcher for times 1,...,k-1. For each node in our network we define three attributes: (i) *Distance*(t,j), which is a lower bound on the total reward incurred in following path s during times 1,...,k-1 and following any extention path from cell s(k) at time k to cell j at time t, t > k; (ii) *Pred*(t,j), which indicates the cell in time t-1 in

the path from s(k) at time k to j at time t; and (iii) *Reward*(t,j), which will take the value of P(t,j), given by equation (2.12).

In a Longest Path Problem, we collect a reward in using an arc, not a node. Therefore, we assign the value of *Reward*(t,j) to all arcs leaving cell j at time t. With this assignment, *Distance*(t,j) is calculated considering the search performed up to time t-1.

The first step is to initialize all the attributes, which is performed by the following procedure [Ref. 7, p. 520].

```
PROCEDURE Initialize(s(k), k, IniDist)
1 FOR t := k TO T+1
2     FOR j := 1 TO N
3         Distance(t,j) := -1;
4         Pred(t,j) := 0;
5         Compute Reward(t,j);
6 Distance(s(k), k) = IniDist;
```

The process of **relaxing** a node is accomplished by determining whether $Distance(t,u) + p_u * Reward(t,u)$ is larger than the present value of $Distance(t+1,v)$, for cells u and v. If this is the case, both $Distance(t+1,v)$ and $Pred(t+1,v)$ are updated.

```
PROCEDURE Relax(u, v, t)
1 Dummy := Distance(t,u) + p_u * Reward(t,u);
2 IF Distance(t+1,v) < Dummy
3 THEN  Distance(t+1,v) := Dummy;
4       Pred(t+1,v) := u;
```

We now create a function to find the longest path in the network, given that some path s was followed up to time $k-1$ (k can be equal to one).

```
FUNCTION DAG_Long_Path1(s(k), k, IniDist, Path)
 1 Initialize(s(k),k, IniDist);
 2 FOR t := k TO T
 3   FOR j := 1 TO N
 4         FOR each cell v ∈ C_j
 5               Relax(j, v, t);
 6 Temp := 0;
 7 FOR j := 1 TO N
 8   IF Distance(T+1,j) ≥ Temp
 9   THEN  Temp := Distance(T+1,j);
10         Path(t) := Pred(T+1,j);
11 FOR t := T-1 DOWNTO k
12   Path(t) := Pred(t+1, Path(t+1));
13 Dag_Long_Path1 := Temp;
```

This function will return both the path maximizing $E[D_t]$ and the value of $E[D_t]$ for this path, and will be used in computing a initial solution for the problem (see *Heuristic_1*, Section II.C).

Since the only information necessary for bounding is E[D], no matter with which path this is accomplished, a different version of this function is used in the branch-and-bound procedure, *Dag_Long_Path2*, without the *Pred(t,j)* attribute and with lines 10 to 12 of the pseudocode deleted.

B. A RECURSIVE BRANCH PROCEDURE

First we defined a new function, *Update*, which will update the target distribution after the search performed in the present time period, before advancing in time.

FUNCTION *UpDate(Reward, t, j)*

1 *Dummy* := p_j * *Reward(t,j)*;

2 *Reward(t,j)* := *Reward(t,j)* - *Dummy*;

3 *UpDate* := *Dummy*;

With the help of the function *Dag_Long_Path2*, we can implement a procedure which will perform the task of deciding whether is it necessary to branch one step ahead in time, updating the target distribution after the search on the present time, or to "fathom" the trial path and restart with another path. With the objective of increasing the flexibility of the program in solving problems of different sizes, both in

number of cells and time available, this branch procedure was implemented in a recursive form. This way, whenever it is necessary to advance in time, the procedure calls itself, until the trial path is "fathomed" or it reaches time T.

Let *Reward(t)* be the vector in N-space of *Reward(t,j)* and let *PBest* and *BestPath* be the best value for PD and respective path obtained so far. Let *Cover(t)* be equal to $Q_s(1, t)$, for some trial path s.

```
PROCEDURE Branch(j, t)
 1 IF t < T
 2 THEN Dummy := Dag_Long_Path2(j, t, Cover(t-1))
 3   IF Dummy ≥ PBest
 4   THEN  Update(Reward, t, j);
 5         FOR each v ∈ C_j
 6               Reward(t+1) := Reward(t) · Γ;
 7               Branch(v, t+1);
 8 ELSE Update(Reward, t, j);
 9    IF Cover(t) ≥ PBest
10    THEN  PBest := Cover(t);
11          BestPath := Path;
```

For a better understanding of the procedure *Branch*, suppose we reach line 7 of the pseudocode at time *t*. When we branch ahead in time, all the information contained in local variables (defined only within the procedure) on the present

call of the function is stored in a stack, waiting until the termination of the next call of *Branch*. In this case, another cell from C_j is tested until no more cells are left. In this case, *Branch* moves back in time, to the previous call.

If the problem we are working with is very large, it may be necessary to provide more memory for the stack than is available in the platform. To prevent this, all variables used in *Branch* are defined to be global variables, which are available to all procedures used in the program and will not be stored in the stack.

C. COMMENTS IN USING PASCAL INSTEAD OF FORTRAN

The Eagle-Yee procedure was implemented in FORTRAN. Our goal in implementing this new procedure in Pascal to make use of some of the features available in this language not present in FORTRAN. Among others, we can mention:

- The ability to define recursive procedures, which increases the flexibility of the program, and allows simpler and more compact code;
- Dynamic memory allocation, which, also, increases the flexibility of the code;
- Pointer variables are already defined in the language. This simplifies the construction and operation of linked lists, trees and other dynamic data structures; and

- User defined data types, which enable the creation of more elaborate and intuitive data structures.

D. COMPLEXITY ANALYSIS OF THE ALGORITHM

As any other branch-and-bound procedure, it may be necessary to verify all possible paths before termination, since we may have the situation where all paths are optimal or near optimal. In this worst case scenario, if at most c cells are available to the searcher in each time period, the procedure *Branch* will be executed, using the O-notation, $O(c^T)$ times. Therefore, we can not guarantee a solution in polynomial time. This was already expected, since this problem is known to be NP-Hard (Trummel and Weisinger, [Ref. 8]).

We also identify the higher sensitivity of the algorithm to the number of time periods for search in the problem, as the number of iterations, in the worst case scenario, increases geometrically with T, while its growth is bounded by a polynomial as c increases, for T fixed.

Another interesting characteristic is that the difficulty in solving the problem is independent of N, the number of cells in the problem, depending only on the number of cells in C_j, for each cell j. Therefore, it makes no difference, at the moment, if we are dealing with a grid of c or c^2 cells, for example. This changes when we consider the computations required for computing the target distribution in each time

period, since this distribution is calculated for all cells and requires multiplication of a N-vector by the N x N transition matrix. If the transition matrix is represented in an adjacency list structure, only the non-zero values will be used in the vector multiplication. Since non-zero entries exit only in each row j for those cells in C_j, it is bounded by c. Therefore, the computational effort in finding the target distribution in times 1 to T is O((N)(c)(T)).

Let's consider the running time required for the heuristic proposed in Section II.C. The computation of the target distribution in time periods t,...,T is O((N)(c)(T-t)), or linear in the size of the network, if the transition matrix is represented in an adjacency list structure. The solution of the longest path problem is also linear in the size of the network [Ref. 7, p. 536]. Both computations are executed inside a main loop from times 1 to T. Therefore, the heuristic running time is $O((N)(c)(T^2))$, representing a real improvement in computational time compared to the branch-and-bound procedure. On the other hand, we are not guaranteed to obtain even near-optimal solutions.

Although good results were observed when applying the heuristic on some of the problems in the OR literature, we can identify situations where it might not work so well. These situations will be analyzed in Chapter IV.

IV. TEST RESULTS

In this Chapter we will present some of the results obtained applying the procedure on problems with different characteristics. Section A involves the comparison with the Eagle-Yee procedure.

A. TESTS AGAINST EAGLE-YEE PROCEDURE

In Eagle and Yee [Ref. 1, p. 114], three different 10-time period search problems were solved. These problems involved 3x3, 5x5 and 7x7 grids, with the searcher starting in cell 1 and the target starting in the other corner of the grid (cells 9, 25 and 49, respectively). Figure 2 represents the situation for the 5x5 grid problem.

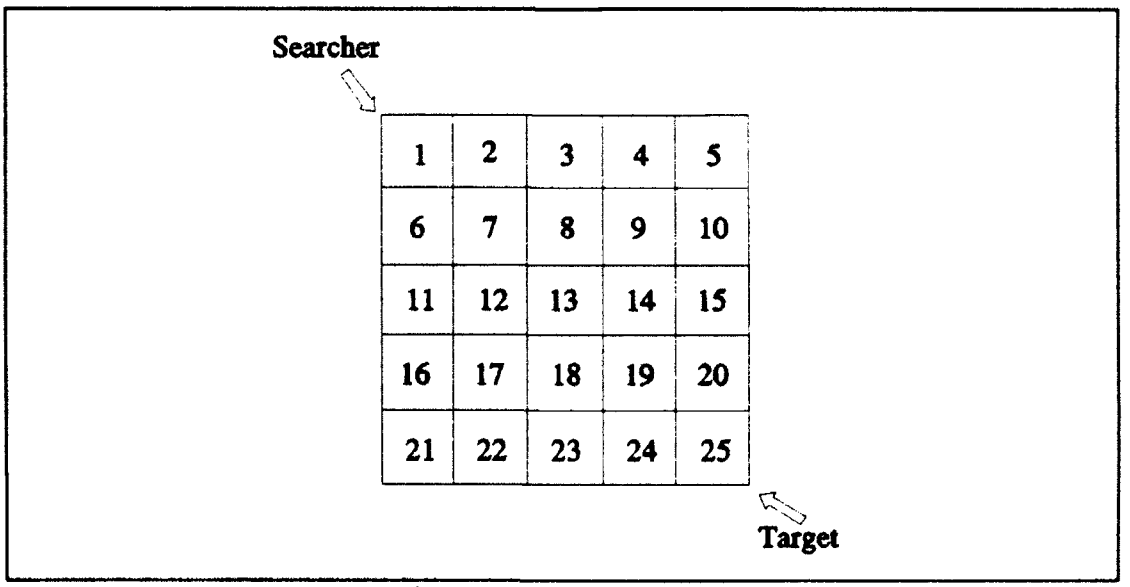

Figure 2 The 5 x 5 Grid

In each time period, the searcher can choose to stay in the same cell or move to an adjacent cell. A cell is adjacent to another if they share a common side. Similarly, the target remains in its present cell with probability 0.4 or moves to one of the adjacent cell, with all adjacent cells being equally likely to be chosen. The probability of detection in each cell is defined to be $1 - e^{-1}$.

In an attempt to compare solutions and running times, both procedures were applied on the three problems listed above. The platform used was the AMDAHL 5990-700A mainframe at the Naval Postgraduate School. The solutions obtained with both procedures were equal. The computational results are summarized in Table I.

As we can observe in Table I, the new procedure usually will require more node evaluations, although it requires less

TABLE I COMPARISON OF COMPUTATIONAL RESULTS

Grid Size	Eagle-Yee		New	
	# Node Evaluations	CPU Time (Seconds)	# Node Evaluations	CPU Time (Seconds)
3 x 3	6,208	17.75	9,660	4.35
5 x 5	3,686	31.37	3,398	3.80
7 x 7	1,945	7.77	2,083	3.46

computational time. An explanation to this result is that the amount of calculations in each branch decision in the new procedure is smaller than in the Eagle-Yee procedure, but the bound used may not be so tight. As the probability of detection increases, the expected number of detections becomes a less efficient bound. This is encountered, for example, when the target probability mass is spread over most of the N cells or the initial distance between searcher and target is small. This explains why the new procedure branches more in the 3x3 problem than in the 7x7. This also affects the efficiency of the Eagle-Yee procedure. As a reference, Table II lists the computational results obtained using a total enumeration program. This program, developed in Pascal, tests all possible paths.

TABLE II COMPUTATIONAL RESULTS WITH TOTAL ENUMERATION

Grid Size	Total Enumeration	
	# Node Evaluations	CPU Time (Seconds)
3 x 3	667,030	43.79
5 x 5	1,582,194	303.90
7 x 7	1,708,266	666.04

Figure 3 is the result of solving the 3x3 and 7x7 grid problems with different values for T, the number of time periods available. The vertical axis scale is logarithmic. Therefore, straight lines represent exponential growth. We can observe that the new procedure, on the observed range, runs faster than the Eagle-Yee procedure. While the running time still grows exponentially with the number of time periods,

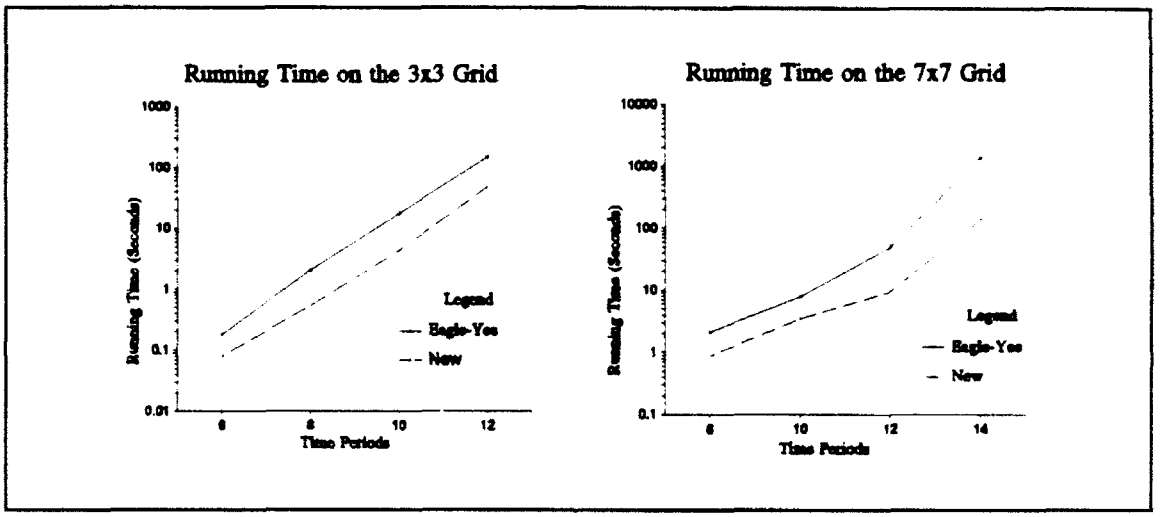

Figure 3 Plot of Running Time vs Time Periods Available

this growth is not so accentuated in the 7x7 grid problem. Recalling from Table I, running times using the new procedure, for both 3x3 and 7x7 grid problems with 10 time periods, were of same order of magnitude. If the number of time periods increases to 12, we will obtain 48.30 and 9.57 seconds, respectively, as running times. If we consider that the number of feasible paths for the searcher increases about 16 times when we increase the number of time periods by 2, we see that

the new procedure is not performing well in the 3x3 grid problem, while it still "filters" much of the growth in the 7x7 grid case.

In Section B we will vary some characteristics of the problems above, like initial distance searcher-target and probability of detection in each cell, to verify how the performance of the new procedure is affected.

B. SENSITIVITY ANALYSIS ON THE NEW PROCEDURE

Since E[D] is an upper bound on PD, if we let the values of p_j, for all cells j, approach zero, our bound would become more tight. In this case the procedure should perform better than with higher values for p_j. Table III was obtained solving the same grid problems above with 12 time periods and using three different values for p_j: 0.3, 0.6 and 0.9. In this table, we can observe that the procedure performs better if the p_j are small.

TABLE III RUNNING TIMES (IN SECONDS) FOR DIFFERENT p_j

Grid Size	Probability of detection in each cell		
	0.3	0.6	0.9
3x3	15.23	43.91	87.41
5x5	10.80	31.83	49.98
7x7	9.33	9.45	10.02

Another characteristic that will affect the performance of the new procedure is the initial distance searcher-target. If the searcher is initially close to the target, the probability of multiple detections is large and E[D] will not be an efficient bound. The same situation was already observed by Caldwell [Ref. 3,p. 46] regarding the Eagle-Yee procedure. Table IV compares the running times of the new procedure when applied on our grid problems, with 12 time periods available, $p_j = 1 - e^{-1}$ and the target initially on the center of the grid and on the opposite corner. Except for the 5x5 grid problem, the running times observed when the target starts in the corner are smaller than when it starts in the center of the grid or closer to the searcher. One explanation of the result observed with the 5x5 grid problem is that the initial solution obtained with the heuristic was better with the target starting in the center.

TABLE IV RUNNING TIME FOR DIFFERENT TARGET INITIAL POSITION

Initial Position of Target	Running Time (Seconds) for Different Grid Sizes		
	3x3	5x5	7x7
Center	73.84	33.97	28.27
Corner	48.30	34.80	9.57

Another characteristic that affects the efficiency of both procedures is the symmetry of the problem. If the problem is symmetric, several optimal or near-optimal paths will be present and the procedure will have to test all. This is the situation we observe in the problems we have been working with. One way to address this situation would be to verify if the problem is symmetric before trying to solve it. This can be accomplished by computing the target distribution for all time periods and checking if the searcher initial position would yield symmetry. If this is the case, we could solve the problem only to one general direction of motion, clockwise or counter-clockwise. We would still guarantee optimality and would require less computational effort.

The last characteristic of the search problem we will test in this section is the number of cells in C_j. In Section III.C we observed that the size of the problem is bounded by a polynomial if we fix the number of time periods and increase the number of reachable cells in each time period. So, we change our grid problems to include as adjacent the cells that share a corner with another. Therefore, diagonal moves are feasible. In Table V we compare running times for the 10 time period grid problems when diagonal moves are allowed. Two important aspects have to be considered when comparing the results: (i) The maximum number of adjacent cells almost

double when we allow diagonal moves, from 5 to 9; and (ii) The initial distance searcher-target decreases. The combined action of these changes causes a large variation on running times. But if we consider that the size of our problems increased, approximately, $(9/5)^{10} \approx 357$ times, we can see that the procedure still performs efficiently.

TABLE V RUNNING TIMES WITHOUT/WITH DIAGONAL MOVEMENT

Grid Size	Running Times (Seconds) Without/With Diagonal Moves	
	No	Yes
3 x 3	4.35	100.01
5 x 5	3.80	30.97
7 x 7	3.46	35.84

In this section, we were able to verify that the efficiency of the new procedure is higher when solving problems with large number of cells. In Section C, we will test the new procedure on larger problems.

C. SOLVING LARGER PROBLEMS

In Section A, we observed that the new procedure is more efficient when the target probability mass is spread over a large number of cells. Therefore, we would expect to obtain relatively fast solutions to problems involving larger grids.

In this section, we will test the procedure on problems involving larger grids or more time periods available.

The first problem to be solved will have almost the same structure as those solved in the previous sections. It will be a 15x15 grid problem, with diagonal moves allowed and with the same transition matrix for the target. The searcher will start in cell 1 and the target in the center of the grid, cell 113. There are 15 time periods available for search.

Table VI lists a summary of the results of applying the new procedure to this problem.

In this problem, the paths maximizing PD involve an initial approach to the center of the target probability mass, followed by what resembles a systematic search around the initial position of the target.

We can observe in Table VI that the computational effort

TABLE VI SUMMARY OF RESULTS FOR THE 15x15 GRID PROBLEM

E[D]		0.252615	
PD given by Heuristic_1		0.196377	
Maximum PD		0.197461	
Running Time (Seconds)	1248.49	# Node Evaluations	104,060
Optimal Paths			
1 17 33 49 65 81 97 113 128 114 98 113 112 128 114 113			
1 17 33 49 65 81 97 113 114 128 112 113 98 114 128 113			

in solving this problem is considerably larger than in the previous problems. We also can observe that the PD obtained with the heuristic differs from the optimal PD by only 0.001. The running time associated with the heuristic, 0.87 seconds, is insignificant when compared with the time required for solving the optmization problem. Therefore, in cases where a fast "good" solution is more important than waiting for an optimal solution, the simple application of the heuristic might be indicated. Figure 4 gives a graphical representation of the first optimal path listed in Table VI.

In Section D we will discuss further the advantages and disadvantages of using the heuristic instead of the branch-and-bound procedure.

The next problem we tried to solve was almost equal to the first 7x7 grid problem presented in Section A, with the difference that 18 time periods would be available for search. An extrapolation from the data in Figure 2 indicated a running time of about 40 minutes. But after 2 hours, the program was still running.

As we can observe, although the branch-and-bound procedure still solves relatively large problems, it may require more computational time than it is available.

Therefore, in cases like these, obtaining a "good" solution with an heuristic might be the only alternative. In Section D we will analyze the results obtained with the

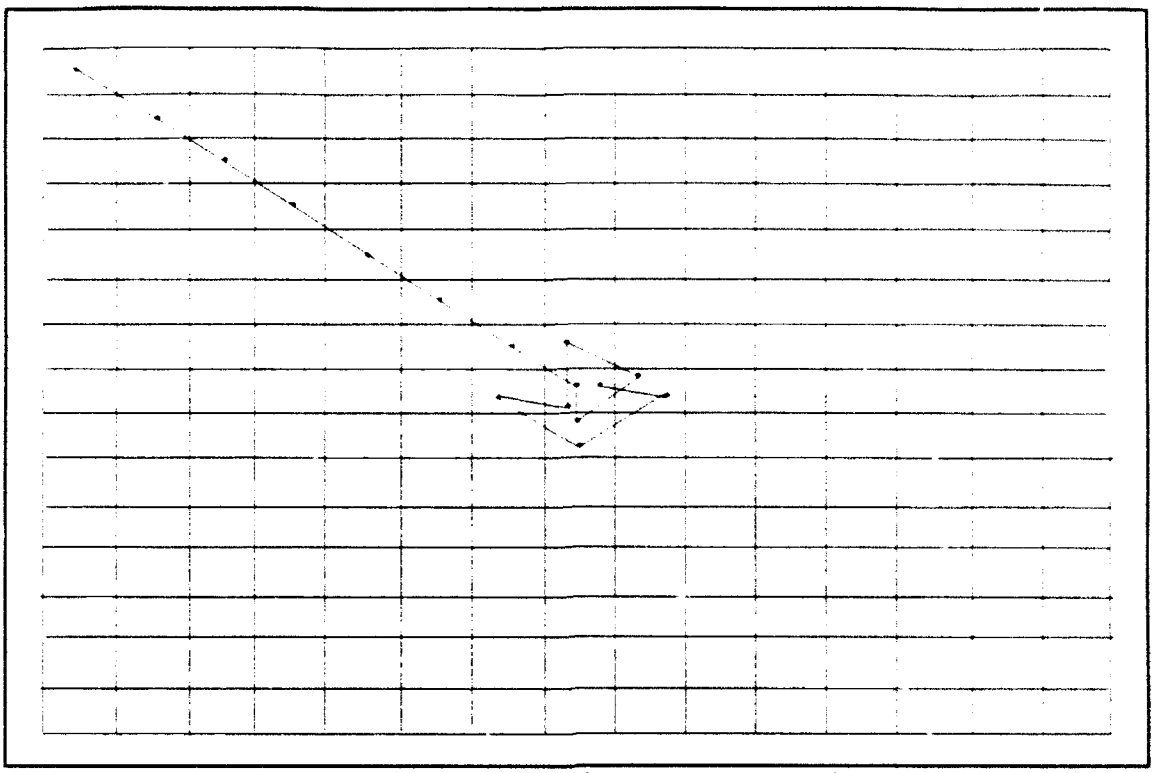

Figure 4 Graphical Representation of an Optimal Path

heuristic and identify some situations were the heuristic might not perform well.

D. PERFORMANCE ANALYSIS OF THE HEURISTIC

Our main objective in proposing an heuristic was to increase the efficiency of the branch-and-bound procedure, by obtaining an initial guess for the solution of the problem. But, as the size of the problem increases, it may only be feasible to apply an heuristic. This is the situation usually encountered when trying to solve problems that are NP-Complete or NP-Hard, like the one we are attempting to solve.

Eagle[Ref. 9] proposed an heuristic using moving horizon policies. In this thesis we propose a new one, which was presented in Section II.C. In this section we will test the performance of the heuristic on different problems.

We will first compare the solution of the heuristic and the optimal solution for some of the problems solved so far. Table VII lists the probability of detection and running time obtained with the heuristic and the branch-and-bound procedure for some of the previous problems.

As we can see from Table VII, the heuristic returned a path that is optimal or near-optimal in each case, while requiring only a fraction of the computational effort used by the branch-and-bound procedure. But it is easy to find examples on which the heuristic will not perform well.

TABLE VII RESULTS OBTAINED WITH HEURISTIC AND B-B PROCEDURE

Grid Size	Time Periods	Branch-and-Bound		Heuristic	
		PD	Run Time (Seconds)	PD	Run Time (Seconds)
3 x 3	10	0.610077	4.35	0.610077	0.014
5 x 5	10	0.358207	3.80	0.358078	0.033
7 x 7	10	0.138220	3.46	0.138220	0.054
3 x 3	12	0.674862	49.36	0.672843	0.019
7 x 7	14	0.314574	141.57	0.314396	0.107
15 x 15	15	0.197461	1248.49	0.196377	0.869

Example 4.1: Suppose, for example, that we have the one dimensional problem depicted in Figure 5. In this case, the target is stationary; initial distribution as shown in Figure 5; p_j is 1.0 for all cells j; and in each time period the searcher can choose to stay in the same cell or move to an adjacent cell. There are three time periods available for search and the searcher starts in cell 4.

In this example, paths (3, 2, 2), (5, 6, 6) and (5, 6, 7) maximize $E[D_s]$. But only one will be stored by the heuristic.

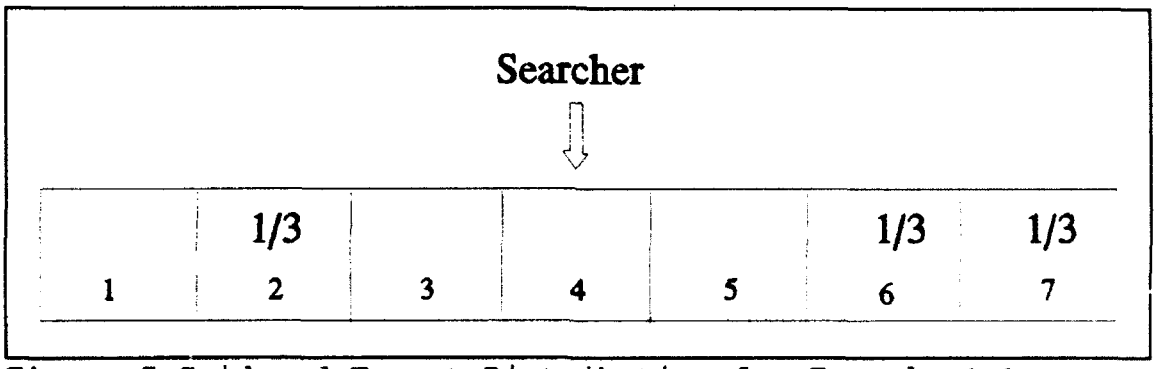

Figure 5 Grid and Target Distribution for Example 4.1

So, depending on how this path is chosen, we may choose cell 3 as the first to be searched. In this case, we will obtain 1/3 for PD, instead of 2/3 obtained with the path (5, 6, 7). Therefore, it is important to create a criterion for breaking ties among paths with the same $E[D_s]$. We may consider as a "tie" if different paths yield results that differ by less than some small positive number, say ϵ.

The criterion we adopted is to choose the path maximizing E[D] with the highest PD. This criterium will detect the situation observed above, but will increase the complexity of the heuristic to $O((N^2)(c)(T^2))$, since it may be necessary to compute PD for N different paths.

We can create another example were the heuristic will not perform well by changing the target initial distribution to 3/7 in cell 2, 2/7 in cells 6 and 7. In this case, the path maximizing $E[D_s]$ is (3, 2, 2) and the path maximizing PD is (5, 6, 7). These paths are completely different. Since the heuristic works by choosing the cell maximizing $E[D_s]$ in each time period, there is no way to prevent this situation. But a slightly modified heuristic may help.

The idea behind this second heuristic is that although the paths maximizing PD may be different from those maximizing $E[D_s]$, they might also yield high values for $E[D_s]$. Suppose that we are verifying which cell will be searched in time t and that we just searched cell j in time period t-1. Instead of choosing the cell in C_j that maximizes $E[D_s]$, we compute PD associated with the paths maximizing $E[D_s]$ for all cells in C_j, with our next cell being the one which yields higher PD. This will not guarantee us the best choice of cell to search, but may correct some "bad" choices from the first heuristic. On the other hand, it also makes "bad" choices, so the best

option is to run both heuristics in tandem. This way, we hope, we will be able to get solutions close to optimal.

In our example, at least, the solution for the second heuristic would be the path (5, 6, 7) and PD = 4/7, the optimal solution.

Since we have to solve the longest path problem for all adjacent cells in each time period, the complexity of this heuristic is $O((N^2)(c^2)(T^2))$. This is still a large improvement when compared with the branch-and-bound procedure. The following is the pseudocode for this heuristic.

```
PROCEDURE Heuristic_2;
1  Path(0) := initial position of searcher;
2  FOR t := 1 TO T
3     Let Path(t-1) be the searcher's cell;
4     FOR all cells j in C_path(t-1)
5           Find path s_j maximizing E[D_s] for times t,...,T,
            given no detections during times 1,...,t-1;
6           Compute PD(j) from s_j;
7     Path(t) := k such that PD(k) := Max_j PD(j);
8     Update(Reward, t, k);
9  Compute PD following Path;
10 Return Path and PD.
```

Our next example will show how this second heuristic performs in a more realistic problem.

Example 4.2: Suppose we have a 10 cells grid as in Figure 6. The searcher starts in cell 1 and the target in cell 5. In each time period the searcher can choose to stay in its present cell or move to an adjacent cell. A cell is adjacent to another if they share a side. Except for cells 2 and 10, in the next time period, the target will stay in its present cell

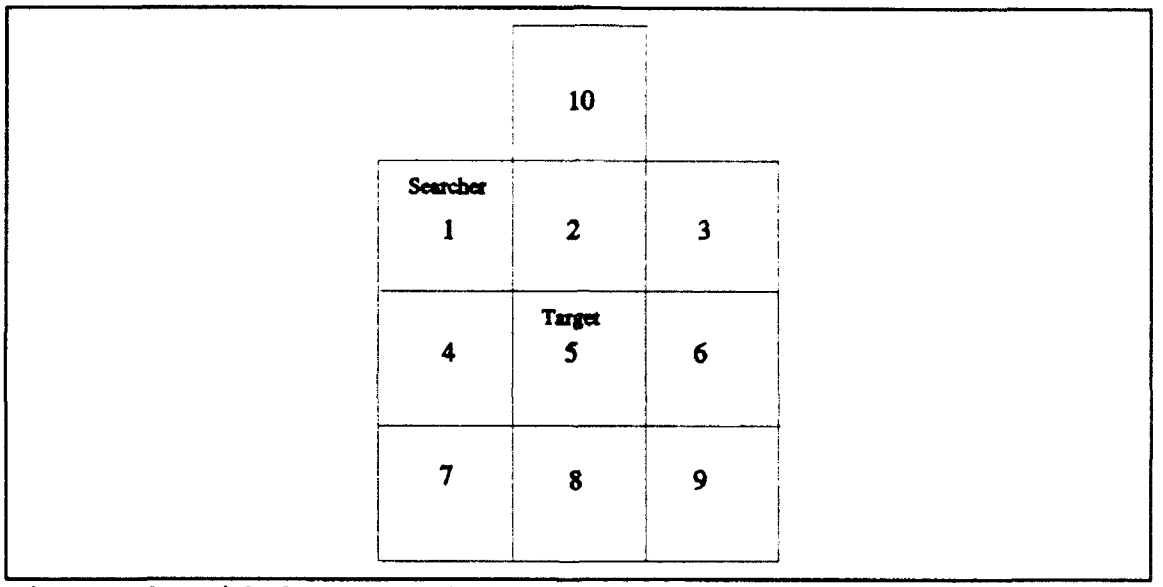

Figure 6 Grid for Example 4.2

with probability 0.2 or move to an adjacent cell, with all adjacent cells equally likely to be chosen. If the target reaches cell 2 it moves with probability 1.0 to cell 10, where it remains. The value for p_j is 1.0, except by cell 10, where $p_{10} = 0.0$. Therefore, if the target reaches cell 10, it will not be detected.

This problem models the situation where the target can evade detection. Using our experience, we can see that if number of time periods, T, is small, the best tactic might be to search different cells in the grid. But if T is larger than some threshold value, the optimal tactic would be to prevent the target from escaping. Table VIII lists the results of solving this problem for T = 15.

As we can observe from the results, our first heuristic fails in preventing the target from evading undetected. This happens because the path maximizing E[D] is considerably distinct from the path maximizing PD. The second heuristic identified this situation and returned a better path, although not optimal. But the running time for both heuristics was 0.13

TABLE VIII RESULTS FOR EXAMPLE 4.2

E[D]	1.423703
Path	1 1 2 5 8 5 8 5 8 8 8 8 8 8 8 8
PD Heuristic 1	0.750316
Path	1 1 2 5 8 8 5 5 8 8 5 5 8 8 5 5
PD Heuristic 2	0.800887
Path	1 1 2 5 2 2 2 2 2 2 2 2 5 8 8 5
Optimal PD	0.806198
Paths	1 1 2 5 2 2 2 2 2 2 5 5 8 8 5 5 1 2 2 5 2 2 2 2 2 2 5 5 8 8 5 5

seconds, while computing the optimal solution required 112.69 seconds. Therefore, since it is difficult to decide which heuristic to use before trying to solve the problem, the best procedure may be running both heuristics and choosing the path with higher PD.

V. CONCLUSIONS

Our main objective in this thesis was to present a new branch-and-bound procedure for computing optimal search paths. This new procedure has been shown to be efficient when tested against the Eagle-Yee procedure. We can not conclude that any of these two procedures is better than the other for all possible applications, but the new procedure required less computational effort than the Eagle-Yee procedure in all problems solved in this thesis. We were also able to verify which characteristics of a problem might affect the efficiency of the new procedure. As a general guideline, it will perform better if PD is small. In this case, the expected number of detections is a tighter upper bound.

Although we were able to implement the branch-and-bound procedure, we observe that as the size of the problem increases, the computational effort required in solving it grows extremly fast.

We also proposed two heuristics, both based in computing paths maximizing the expected number of detections, E[D].

In the first heuristic, the cell to be searched in time period k is the one that maximizes the expected number of detections for t = k,...,T. In the second heuristic, if cell j was searched at time k-1, the cell to be searched in time

period k is the cell in C_j for which the path maximizing the expected number of detections yields the larger PD. The values for PD obtained with the combination of these two heuristics were close to the optimal PD, but required only a fraction of the computational effort required for the branch-and-bound procedure.

Therefore, if a fast approximate solution is more important than an optimal solution, using one of the heuristics may be indicated.

Since it may be impractical to identify which heuristic will perform better in each different scenario, we suggest the use of both.

APPENDIX

```
PROGRAM Find_Path;
{-------------------------------------------------------------------------
 This program uses a branch-and-bound procedure, with the bound given
by the expected number of detections, to identify the optimal paths
for the path constrained search problem.
 It includes two heuristics for finding "good" initial solutions.

 Author: LCDR Gustavo Martins, Brazilian Navy
 Date: 02/26/93
-------------------------------------------------------------------------}
Uses DOS;  {Unit DOS is used for timming}

    CONST MTime    =      25;   {Maximum # of time steps}
          MCell    =     225;   {Maximum # of cells}
          Eps      =  0.0001;   {Small positive number}
          Alpha    = 0.63212;   {Default prob. of detection}

    TYPE PAdj = ^TAdj;
         PMar = ^TMar;
         PMR  = ^MatrixR;
         PMI  = ^MatrixI;
         TAdj = RECORD
                     Head : INTEGER;
                     Next : PAdj;
                END;
         TMar = RECORD
                     Node : INTEGER;
                     Pij  :    REAL;
                     Next :    PMar
                END;

         AdjList  = ARRAY[1..MCell] OF PAdj;
         VectorR  = ARRAY[1..MCell] OF REAL;
         VectorI  = ARRAY[1..MCell] OF INTEGER;
         PathI    = ARRAY[0..MTime] OF INTEGER;
         PathR    = ARRAY[0..MTime] OF REAL;
         MatrixR  = ARRAY[1..MTime + 1] OF VectorR;
         MatrixI  = ARRAY[1..MTime + 1] OF VectorI;
         MarkovR  = ARRAY[1..MCell] OF PMar;

    VAR  Reward           :     PMR;  {Matrix of rewards associated
                                         with each node}
         Cell             : AdjList;  {Adjacency matrix in linked
                                         list form}
         I, K, T, J       : INTEGER;  {Dummy variables}
         LTime, NCell     : INTEGER;  {# of time steps and cells
                                         in the input problem}
         Searcher         : INTEGER;  {Initial position of searcher}
         PP, TP           :    PAdj;  {Dummy pointers}
         DataIn, DataOut  :    TEXT;  {Input and output files}
         Ini, PD          : VectorR;  {Ini: Initial distribution
                                              of target
                                         PD:  Probability of detection
```

```
                                              for each cell}
        Mat              : MarkovR;  {Transition matrix in linked
                                      list form}
        OpPath, Path     :   PathI;  {Vectors for recording the path
                                      of the searcher}
        BPath            :   PathI;  {Idem}
        Sol              :    REAL;  {Present best solution}
        Cover            :   PathR;  {Vector that keeps track of
                                      the probability mass covered
                                      at each step}
        Counter          : INTEGER;  {Count number of iterations}
        Temp, Dif        :    REAL;  {Used to control running time}
        Hour, Minute     :    WORD;  {Idem}
        Second, Hundred  :    WORD;  {Idem}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

PROCEDURE ReadData(VAR DataIn :    TEXT;
                   VAR Mat    : MarkovR;
                   VAR Cell   : AdjList);
{------------------------------------------------------------------------------
 This procedure reads the input file.
 The input file must be on following format:
     NCell LTime Searcher

     Ini  (Initial distribution, as a list: Cell Prob Cell Prob...)

     1,  if cells have prob of detect different of 1 - EXP(-1);
     0, otherwise

     Transition Matrix, with each line corresponding to one cell:
          First two values are Cell  and Prob. detect. in that cell
          If you use  0 above, you can use 0.0 as Prob. detect.
          then  NextCell Prob NextCell Prob NextCell Prob...

    Adjacency list for the searcher:
     Cell NextCell NextCell...
------------------------------------------------------------------------------}

    VAR I, J, K  : INTEGER;
        PP1      :    PAdj;
        PP2      :    PMar;
        Prob, Pj :    REAL;
        Dif_PI   : BOOLEAN;

BEGIN {Procedure ReadData}
    READLN(DataIn, NCell, LTime, Searcher);
    READLN(DataIn);
    FOR I := 1 TO NCell DO
    BEGIN
        Ini[I]  := 0.0;
        Cell[I] := NIL;
        Mat[I]  := NIL
    END;
    WHILE NOT EOLN(DataIn) DO
    BEGIN
        READ(DataIn, K, Prob);
        Ini[K] := Prob
```

```
        END;
        READLN(DataIn);
        READLN(DataIn);
        READLN(DataIn, K);
        Dif_PD := (K = 1);
        IF NOT Dif_PD THEN
           FOR I := 1 TO NCell DO
               PD[I] := Alpha;
        READLN(DataIn);
        FOR I := 1 TO NCell DO
        BEGIN
            READ(DataIn, K, Pj);
            WHILE NOT EOLN(DataIn) DO
            BEGIN
                IF Dif_PD THEN PD[I] := Pj;
                READ(DataIn, J, Prob);
                PP2 := Mat[J];
                New(Mat[J]);
                Mat[J]^.Node :=    I;
                Mat[J]^.Pij  := Prob;
                Mat[J]^.Next :=  PP2
            END
        END;
        READLN(DataIn);
        FOR I := 1 TO NCell DO
        BEGIN
            READ(DataIn, K);
            WHILE NOT EOLN(DataIn) DO
            BEGIN
                READ(DataIn, J);
                PP1  := Cell[I];
                NEW(Cell[I]);
                Cell[I]^.Head :=   J;
                Cell[I]^.Next := PP1
            END
        END;
    END;  {Procedure ReadData}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

PROCEDURE VecMatProd(VecIn      : VectorR;
                     Mat        : MarkovR;
                     VAR VecOut : VectorR);
{-------------------------------------------------------------------------
 This procedure returns the vector obtained by postmultipling the row
 vector VecIn by the matrix Mat, where mat is in linked list form.
-------------------------------------------------------------------------}

    VAR I  : INTEGER;
        TP :    PMar;

BEGIN {Procedure VecMatProd}
    FOR I := 1 TO NCell DO
    BEGIN
        VecOut[I] := 0;
        TP := Mat[I];
        WHILE TP <> NIL DO
        BEGIN
```

```
            VecOut[I] := VecOut[I] + VecIn[TP^.Node] * TP^.Pij;
            TP := TP^.Next
        END
    END;
END;  {Procedure VecMatProd}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

FUNCTION UpDate(Node, Time  : INTEGER;
                VAR FReward :     PMR): REAL;
{-------------------------------------------------------------------------}
 This function updates the target distribution after the search in
cell Node in time period Time, and returns the probability of initial
detection in that time period.
{-------------------------------------------------------------------------}

    VAR Temp : REAL;

BEGIN {Function UpDate}
    Temp  := PD[Node] * FReward^[Time, Node];
    FCost^[Time, Node] := FReward^[Time, Node] - Temp;
    UpDate := Temp;
END;  {Function UpDate}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

FUNCTION Dag_Long_Path1(Source, Step : INTEGER;
                        IniDist      :    REAL;
                        Reward       :     PMR;
                        VAR Expect   :    REAL): REAL;
{-------------------------------------------------------------------------}
 This Function computes the Longest Path, given the present position,
the present time step and the matrix with the reward associated with
each node.
 The output is the Longest Path, the total reward (expected number of
 detections) and the probability of detection associated with this
 path.
{-------------------------------------------------------------------------}

    VAR Connected  : BOOLEAN;  {Guarantee network constraints}
        I, T       : INTEGER;  {Index variables}
        PP         :    PAdj;  {Pointer}
        Temp, Best :    REAL;  {Dummy variable}
        Parent     :     PMI;  {Predecessor attribute}
        Distance   :     PMR;  {Distance attribute}
        BReward    :     PMR;  {Reward attribute}
        TPath      :   PathI;  {Dummy path variable}

{++++++++++++++++++++++++++++}

PROCEDURE Initialize_1(Source, Step : INTEGER;
                       IniDist      :    REAL;
                       VAR CReward  :     PMR);

    VAR I, T      : INTEGER;

BEGIN {Procedure Initialize_1}
    FOR T := Step To LTime + 1 DO
```

```
        FOR I := 1 TO NCell DO
        BEGIN
            Distance^[T, I]    := -1.0;
            Parent^[T, I]      :=    0
        END;
    Distance^[Step, Source] := IniDist;
    FOR T := Step  TO LTime-1  DO
        VecMatProd(CReward^[T], Mat, CReward^[T+1]);
END;  {Procedure Initialize_1}

{*****************************}

PROCEDURE Relax_1(U, V, T : INTEGER);

    VAR R1 : REAL;

BEGIN {Procedure Relax}
    R1 := Distance^[T, U] + PD[U]*BReward^[T, U];
    IF Distance^[T + 1, V] < R1 THEN
    BEGIN
        Distance^[T + 1, V] := R1;
        Parent^[T + 1, V] := U
    END;
END;  {Procedure Relax}

{+++++++++++++++++++++++++++}

BEGIN {Procedure Dag_Long_Path1}
    NEW(Distance);
    NEW(Parent);
    New(BReward);
    TPath := Path;
    BReward^[Step] := Reward^[Step];
    Initialize_1(Source, Step, IniDist, BReward);

    FOR T := Step TO LTime DO
        FOR I := 1 TO NCell DO
        BEGIN
            PP := Cell[I];
            Connected := Distance^[T, I] >  - Eps;
            WHILE (PP <> NIL) AND Connected DO
            BEGIN
                Relax_1(I, PP^.Head, T);
                PP := PP^. Next
            END
        END;

    Expect := - 1.0;
    FOR I := 1 TO NCell DO
          IF Distance^[LTime + 1, I] > Expect THEN
             Expect  := Distance^[LTime + 1, I];
    Best := -1.0;
    FOR I := 1 TO NCell DO
        IF Expect = Distance^[LTime + 1, I] THEN
        BEGIN
            TPath[Ltime] := Parent^[LTime + 1, I];
            FOR T := LTime - 1 DOWNTO Step DO
                TPath[T] := Parent^[T + 1, TPath[T + 1]];
```

```
            Temp := 0.0;
            BReward^[Step] := Reward^[Step];
            FOR K := Step TO LTime DO
            BEGIN
                Temp := Temp + UpDate(TPath[K], K, BReward);
                VecMatProd(BReward^[K], Mat, BReward^[K+1]);
            END;
            IF Temp > Best THEN
            BEGIN
                Best := Temp;
                Path := TPath
            END
        END;
    Best := Best + IniDist;
    DISPOSE(Distance);
    DISPOSE(Parent);
    DISPOSE(BReward);
    Dag_Long_Path1 := Best;
END; {Function Dag_Long_Path1}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

FUNCTION Dag_Long_Path2A(Source, Step : INTEGER;
                         IniDist      :    REAL;
                         VAR DReward    :     PMR): REAL;
{-------------------------------------------------------------------------
 This Function computes the Longest Path, given the present position,
the present time step and the matrix with the reward associated with
each node.
 The output is the total reward (expected number of detections)
associated with this path.
-------------------------------------------------------------------------}

    VAR Connected : BOOLEAN;
        I, T      : INTEGER;
        PT        :    PAdj;
        Temp      :    REAL;
        Distance  :     PMR;

{++++++++++++++++++++++++++++}

PROCEDURE Initialize_2A(Source, Step : INTEGER;
                        FDist        :    REAL);

    VAR I, T      : INTEGER;

BEGIN {Procedure Initialize_2A}
    FOR T := Step To LTime + 1 DO
        FOR I := 1 TO NCell DO
            Distance^[T, I]    := - 1;
    Distance^[Step, Source]    := FDist;
    FOR T := Step TO (LTime - 1) DO
        VecMatProd(DReward^[T], Mat, DReward^[T + 1]);
END;  {Procedure Initialize_2A}

{*****************************}

PROCEDURE Relax_2A(U, V, T : INTEGER);
```

```
    VAR R1 : REAL;

BEGIN {Procedure Relax_2A}
    R1 := Distance^[T, U] + PD[U] * DReward^[T, U];
    IF Distance^[T + 1, V] < R1 THEN
        Distance^[T + 1, V] := R1;
END;  {Procedure Relax_2A}

{++++++++++++++++++++++++++}

BEGIN {Procedure Dag_Long_Path2A}
    NEW(Distance);
    Initialize_2A(Source, Step, IniDist);

    FOR T := Step TO LTime DO
        FOR I := 1 TO NCell DO
        BEGIN
            PT := Cell[I];
            Connected := Distance^[T, I] >  - Eps;
            WHILE (PT <> NIL) AND Connected DO
            BEGIN
                Relax_2A(I, PT^.Head, T);
                PT := PT^. Next
            END
        END;

    Temp := 0.0;
    FOR I := 1 TO NCell DO
          IF Distance^[LTime + 1, I] > Temp THEN
            Temp  := Distance^[LTime + 1, I];
    DISPOSE(Distance);
    Dag_Long_Path2A := Temp;
END; {Function Dag_Long_Path2A}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

PROCEDURE InitialSolution(VAR Sol   :  REAL;
                          VAR BPath : PathI);
{-------------------------------------------------------------------|
| This procedure computes the maximum expected number of detections  |
|that can be obtained with any of the feasible paths. It also uses two |
|heuristics to obtain a "good" initial solution.                     |
|-------------------------------------------------------------------}

    VAR K             : INTEGER;
        TCover, Res   :    REAL;
        TSol, Expect  :    REAL;
        TPath         :   PathI;

BEGIN {Procedure InitialSolution}
    Reward^[1]  := Ini;
    OpPath[0]  := Searcher;
    BPath[0]   := Searcher;
    Path[0]    := Searcher;
    TPath[0]   := Searcher;
    Sol     := 0.0;
    TSol    := 0.0;
    PP := Cell[BPath[0]];
```

```
WHILE PP <> NIL DO
BEGIN
    Res := Dag_Long_Path1(PP^.Head, 1, 0.0, Reward, Expect);
    IF Expect >= TSol THEN
    BEGIN
        Sol    :=    Res;  {PD associated with BPath}
        TSol   := Expect;  {Max expected number of detections}
        BPath  :=   Path   {Best path so far}
    END;
    PP := PP^.Next
END;

WRITELN(DataOut, 'Path maximizing E[D]: ');
FOR K := 0 TO LTime DO
    WRITE(DataOut, BPath[K]:4);
WRITELN(DataOut);
WRITELN(DataOut, '  E[D] = ', TSol:10:6);
WRITELN(DataOut, '  PD   = ', Sol:10:6);
WRITELN(DataOut);

Path[1] := BPath[1];
TCover := 0.0;
FOR K := 1 TO LTime DO  {Executes heuristic 1}
BEGIN
    TCover := TCover + UpDate(Path[K], K, Reward);
    VecMatProd(Reward^[K], Mat, Reward^[K+1]);
    TSol   := Dag_Long_Path1(Path[K], K, 0.0, Reward, Expect)
END;

IF TCover > Sol THEN
BEGIN
    Sol   := TCover;
    BPath :=   Path
END;

WRITELN(DataOut, 'Solution of heuristic_1: ');
FOR K := 0 TO LTime  DO
    WRITE(DataOut, Path[K]:4);
WRITELN(DataOut);
WRITELN(DataOut, ' PD  =  ', TCover:10:6);
WRITELN(DataOut);

TCover := 0.0;
FOR K := 1 TO LTime DO   {Executes heuristic 2}
BEGIN
    PP  := Cell[TPath[K-1]];
    TSol := 0.0;
    WHILE PP <> NIL DO
    BEGIN
        Res  := Dag_Long_Path1(PP^.Head, K, 0.0, Reward, Expect);
        IF Res >= TSol THEN
        BEGIN
            TSol     := Res;
            TPath[K] := PP^.Head
        END;
        PP := PP^.Next
    END;
    TCover := TCover + UpDate(TPath[K], K, Reward);
```

```
        VecMatProd(Reward^[K], Mat, Reward^[K+1]);
    END;

    WRITELN(DataOut, 'Solution of heuristic_2: ');
    FOR K := 0 TO LTime  DO
        WRITE(DataOut, TPath[K]:4);
    WRITELN(DataOut);
    WRITELN(DataOut, ' PD =  ', TCover:10:6);
    WRITELN(DataOut);

    IF TCover > Sol THEN
    BEGIN
        Sol   := TCover;
        BPath := TPath
    END;

    WRITELN(DataOut, 'Initial Solution :');
    FOR K := 0 TO LTime  DO
        WRITE(DataOut, BPath[K]:4);
    WRITELN(DataOut);
    WRITELN(DataOut, ' PD =  ', Sol:10:6);
    WRITELN(DataOut);
END;  {Procedure InitialSolution}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

PROCEDURE Branch(Node, Time : INTEGER);
{-------------------------------------------------------------------------
 This is a recursive procedure which will perform the branch-and-bound
steps of the program. At each branch decision, an upper bound on the
probability of detection is computed. The procedure will call itshelf
and branch ahead in time if this bound is greater than the previous
best result. Otherwise, it will "fathom".
 In the last time step, it will output the path and PD.
-------------------------------------------------------------------------}

    VAR PP       :    PAdj;
        T, N     : INTEGER;
        RR, Gone :    REAL;

BEGIN {Procedure Branch}
    IF Time < LTime THEN
    BEGIN
        RR := Dag_Long_Path2A(Node, Time, Cover[Time-1], Reward);
        Counter := Counter + 1;
        IF RR >= Sol THEN    {Compare bound with best result}
        BEGIN
            Path[Time]  := Node;
            Cover[Time] := Cover[Time-1] + UpDate(Node, Time, Reward);
            PP := Cell[Node];
            WHILE PP <> NIL DO
            BEGIN
                VecMatProd(Reward^[Time], Mat, Reward^[Time + 1]);
                N := PP^.Head;
                Branch(N, Time + 1);  {Branch ahead in time}
                PP := PP^.Next
            END
        END
```

```
    END
    ELSE
    BEGIN
        Path[LTime] := Node;
        Cover[LTime] := Cover[LTime-1] + UpDate(Node, LTime, Reward);
        IF Cover[LTime] >= Sol THEN
        BEGIN
            WRITE(DataOut, 'New Best Sol: ');
            FOR T := 0 TO LTime DO
                WRITE(DataOut, Path[T]:4);
            WRITELN(DataOut, Cover[LTime]:10:6);
            Sol    := Cover[LTime];
            OpPath := Path
        END
    END;
END;  {Procedure Branch}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

PROCEDURE ComputeTime;
{--------------------------------------------------------------------------}
| This procedure computes the time used by each algorithm.                 |
| USES DOS.                                                                |
{--------------------------------------------------------------------------}

BEGIN {Procedure ComputeTime}
    GetTime(Hour, Minute, Second, Hundred);
    Dif := (Hour * 60) + Minute + (Second / 60) - Temp;
END;  {Procedure ComputeTime}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

PROCEDURE RecordTime;
{--------------------------------------------------------------------------}
| This procedure records the starting time of each algorithm.              |
| USES DOS.                                                                |
{--------------------------------------------------------------------------}

BEGIN {Procedure RecordTime}
    GetTime(Hour, Minute, Second, Hundred);
    Temp := (Hour * 60) + Minute + (Second / 60);
END;  {Procedure RecordTime}

{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}

BEGIN {Program Find_Path}
    ASSIGN(DataIn, 'C:\THESIS\THESISI1.txt');
    ASSIGN(DataOut, 'C:\THESIS\THESISO1.txt');

    RESET(DataIn);
    ReadData(DataIn, Mat, Cell);
    CLOSE(DataIn);

    REWRITE(DataOut);
    WRITELN(DataOut, 'Number of cells in the problem: ', NCell:4);
    WRITELN(DataOut);
    WRITELN(DataOut, 'Number of time steps in the problem: ', LTime:4);
    WRITELN(DataOut);
```

```
      RecordTime;
      NEW(Reward);
      InitialSolution(Sol, BPath);
      Counter  := 0;
      Cover[0] := 0.0;
      PP := Cell[Searcher];
      WHILE PP <> NIL DO
      BEGIN
           Reward^[1] := Ini;
           Branch(PP^.Head, 1);
           PP := PP^.Next
      END;
      DISPOSE(Reward);
      ComputeTime;

      WRITELN(DataOut);
      WRITELN(DataOut, 'Optimal Solution :');
      FOR J := 0 TO LTime  DO
          WRITE(DataOut, OpPath[J]:4);
      WRITELN(DataOut);
      WRITELN(DataOut, ' PD =  ', Sol:10:6);
      WRITELN(DataOut);
      WRITELN(DataOut, 'Number of node evaluations: ', Counter:7);
      WRITELN(DataOut, 'Computation time in minutes:  ', Dif:10:6);
      CLOSE(DataOut);
END. {Program Find_Path}
```

LIST OF REFERENCES

1. Eagle, J. and Yee, J., "An Optimal Branch-and-Bound Procedure for the Constrained Path, Moving Target Search Problem", *Operations Research*, v. 38, no. 1, pp. 110-114, January-February 1990.

2. Frank, M. and Wolfe, P., "An Algorithm for Quadratic Programming", *Naval Research Logistics Quarterly*, v. 3, nos. 1-2, pp.95-110, March-June 1956.

3. Caldwell, J., "Investigation and Implementation of an Algorithm for Computing Optimal Search Paths", *Naval Postgraduate School*, Monterey, CA, September 1987.

4. Stewart, T. J., "Search for a Moving Target When Searcher Motion is Restricted", *Computers and Operations Research*, v. 6, pp.129-140, 1979.

5. Eagle, J. and Washburn, A., "Cumulative Search-Evasion Games", *Naval Research Logistics*, v. 38, no. 4, pp. 495-510, August 1991.

6. Washburn, A., "Search and Detection", *ORSA Books*, Arlington, VA, 1981.

7. Cormen, T., Leiserson, C., and Rivest, R., "Introduction to Algorithms", The MIT Press,Cambridge, MA, 1990.

8. Trummel, K. E., and Weisinger, J. R., "The Complexity of the Optimal Searcher Path Problem", *Operations Research*, v. 34, pp. 324-327, 1986.

9. Eagle, J., "The Approximate Solutions of a Simple Constrained Search Path Moving Target Problem Using Moving Horizon Policies", Technical Report NPS55-84-009, *Naval Postgraduate School*, Monterey, CA.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Superintendent Attn: Library, Code 1424 Naval Postgraduate School Monterey, California 93943-5000	2
3.	Professor James N. Eagle, Code OR/Er Naval Postgraduate School Monterey, California 93943-5000	2
4.	Professor Craig W. Rasmussen, Code MA/Ra Naval Postgraduate School Monterey, California 93943-5000	2
5.	Professor Alan R. Washburn, Code OR/Ws Naval Postgraduate School Monterey, California 93943-5000	1
6.	Comando de Operações Navais Attn: Brazilian Naval Commission 4706 Wisconsin Ave., N. W. Washington, D.C. 20016	1
7.	Brazilian Naval Commission 4706 Wisconsin Ave., N. W. Washington, D.C. 20016	1
8.	Centro de Analises de Sistemas Navais Attn: Brazilian Naval Commission 4706 Wisconsin Ave., N. W. Washington, D.C. 20016	1
9.	LCDR Gustavo H. A. Martins Attn: Brazilian Naval Commission 4706 Wisconsin Ave., N. W. Washington, D.C. 20016	2
10.	LT Almir G. Santos, SMC 1527 Naval Postgraduate School Monterey, California 93943-5000	1